IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

INVENTOR(S): **L. Scott Rich, Ritchard L. Schacher and Daniel Berg**

APPLICATION NO. **09/824,614**

FILED: **April 2, 2001**

TITLE: **METHOD AND APPARATUS FOR EFFICIENTLY REFLECTING COMPLEX SYSTEMS OF OBJECTS IN XML DOCUMENTS**

CERTIFICATE OF MAILING/FACSIMILE

I hereby certify that this correspondence, along with any paper indicated as being enclosed, are being deposited with the United States Postal Service as first-class mail, postage pre-paid, on July 6, 2005 in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

Date

Diane A. Sears

Commissioner for Patents
Post Office Box 1450
Alexandria, VA 22313-1450

DECLARATION OF L. SCOTT RICH, RITCHARD L. SCHACHER AND DANIEL BERG UNDER 37 C.F.R. §1.131

Sir:

We, L. Scott Rich, Ritchard L. Schacher and Daniel Berg, hereby declare as follows:

1. We are the inventors of the subject matter claimed in the above-identified patent application.

2. We have been informed that U.S. Patent Publication No. 2002/0128734 to Dorsett, Jr. (hereinafter Dorsett) has been cited as prior art by the United States Patent and Trademark Office with respect to the above-identified patent application. We also have been informed that the earliest claimed filing date of Dorsett is January 5, 2001.

3. This declaration is to establish conception and reduction to practice of the invention claimed in claims 1-15 of the above-identified application in the United States prior to January 5, 2001.

4. Prior to January 5, 2001, we had conceived of and developed (i.e., reduced to practice) a working software prototype program that met all of the limitations of claims 1-15 of our patent application, as more particularly detailed below.

5. Document A attached hereto is memo prepared by named inventor Scott Rich concerning the development of the invention disclosed and claimed in the above-identified patent application. The date appearing on the original of this document has been redacted from the copy attached hereto. However, the date is prior to January 5, 2001 and accurately reflects a date on which the document existed.

6. Document B attached hereto is a memo prepared by named inventor Scott Rich further concerning the development of the invention disclosed and claimed in the above-identified patent application. The date appearing on the original of this document has been redacted from the copy attached hereto. However, the date is after the date of Document A, but also prior to January 5, 2001 and accurately reflects a date on which the document existed.

7. Document C attached hereto is an e-mail sent by named inventor Richard L. Schacher further concerning the development of the invention disclosed and claimed in the above-identified patent application. The date appearing on the original of this document has been redacted from the copy attached hereto. However, the date is after the date of Document B, but also prior to January 5, 2001 and accurately reflects a date on which the document existed. Note that Document C contains actual code for implementing the invention and therefore evidences actual reduction to practice of the invention.

8. Document D attached hereto is an e-mail sent by named inventor Richard L. Schacher further concerning the development of the invention disclosed and claimed in the above-identified patent application. The date appearing on the original of this document has been redacted from the copy attached hereto.

However, the date is after the date of Document C, but also prior to January 5, 2001 and accurately reflects a date on which the document existed. Note that Document D contains actual code for implementing the invention and therefore evidences actual reduction to practice of the invention.

9. The present invention as claimed relates to software for efficiently exchanging data between two computer application programs or between a resource library and an application program. In accordance with the invention. XMI documents for transporting the data between the two applications are built on-the-fly rather than being stored in memory. Further, when a resource library or application program receives a request for an object, the resource library creates the resource to which that object corresponds, but does not populate that resource. Next, the resource library populates the resource with only the object(s) requested. Then the resource is returned to the requesting application program.

10. We conceived and reduced to practice the invention claimed in claims 1-15 of the above-identified patent application prior to January 5, 2001. Documents A and B demonstrate conception of the invention. Documents C and D include actual code implementing the invention and, therefore, constitute actual reduction to practice.

11. The attached claim table maps the claims element-by-element to the evidence submitted herewith. In the table, the left-hand column shows each claim element in the claims and the right-hand column indicates the section of the evidence that demonstrates that we had possession of that element of the invention at that time. All citations in the right-hand column are given by document number, page number(s) within the document, and line number(s) within the document, except as otherwise indicated. The document numbers, page numbers, and line numbers did not appear in the original documents, but were added for ease of reference in connection with this declaration.

12. As the persons signing below, we each hereby declare that all statements made herein of my own knowledge are true and that all statements

made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment or both, under Section 1001 of Title 18 of the United States Code, and that such willful statements may jeopardize the validity of the application or any patent issued thereon.

6/27/2005
Dated

L. Scott Rich
L. Scott Rich

6/27/2005
Dated

Richard L. Schacher
Ritchard L. Schacher

6/29/2005
Dated

Daniel Berg
Daniel Berg

CLAIM TABLE

	Claim Element	Corresponding disclosure in submitted evidence
1	1. A method for exchanging objects between two computing entities in an object-oriented programming environment using a transport mechanism in which said data units are contained in files, each file defining a resource, each resource designed to contain a plurality of particular ones of said objects, said method comprising the steps of: (1) providing a resource factory for building resources, said factory including a plurality of software modules for building resources from a data source, each said software module designed to build a resource of a particular type; (2) responsive to a request for an object from a first computing entity, selecting a software module for building a resource of the type to which said object corresponds; (3) building a resource for containing said object using said selected software module, said resource populated with information defining said resource, but not containing said object; (4) inserting said object into said resource; (5) transmitting said resource to said first computing entity using said transport mechanism; and (6) providing to said first computing entity said object.	The concept of claim 1 in its entirety is found in Document 1. See particularly lines 23-36, quoted below.
2	2. The method of claim 1 wherein, in step (4), only said object is inserted in said resource.	"Java (meaning the Java model resource) will poof up a JavaClass (the said object) if needed". Document 1, lines 25-26.

3	3. The method of claim 2 further comprising the
	steps of:
	(7) providing a reflection adapter factory for populating objects within resources, said factory adapted to provide software modules for populating objects, each said software module designed for an environment corresponding to an object;
	(8) responsive to a request for a property of said object, selecting a one of said reflection adapters for the environment of the particular property;
	(9) populating said object with said property; and
	(10) providing to said first computing unit said property.
		/**
		* Perform the appropriate action.
		*/
		public void actionPerformed(IAction action, Window shell) {
		// No-op for now
		IWorkbench w = EJBPlugin.getPluginWorkbench();
		ISolution s = ((Workbench) w).getOpenSolution();
		IJavaModel javaModel = JavaModelManager.getJavaModel(s);
		IJavaProject[] projects = null;
		try {
		projects = javaModel.getJavaProjects();
		} catch (JavaModelException e) {
		System.out.println("no loaded projects");
		return;
		}
		IJavaProject firstProject = projects[0];
		Factory f = FactoryImpl.instance();
		Document d = f.createDocument("testing.xmi");
		((XMIDocumentImpl) d.getXMIDocument()).addAdaptorFactory(new JavaJDOMAdapterFactory(firstProject));
		JavaPackage javaLang = f.createJavaPackage();
		javaLang.setName("java.lang");
		JavaClass string = f.createJavaClass();
		string.setName("String");
		javaLang.addContents(string);
		// DebugSupport.inspect(string);
		
		string.refValue(FactoryImpl.JavaClass_isPublic);
		
		This code from Document 3, lines 5-31 embodies the claimed subject matter.

4	4. The method of claim 3 wherein said object comprises a plurality of properties	The code from Document 3, lines 5-31 quoted immediately above also embodies this claimed subject matter.
5	and step (9) comprises populating said object with all properties of said object that can be reflected.	This is shown in the code found in Document 4, page 5, line 26-page 6, line 20.
6	5. A method for exchanging objects between two computing entities in an object-oriented programming environment using a transport mechanism in which said data units	Same recitation as in claim 1. See row 1 above.

	are contained in files, each file defining a resource, each resource designed to contain a plurality of particular ones of said objects, said method comprising the steps of: (1) providing a resource factory for building resources, said factory including a plurality of software modules for building resources from a data source, each said software module designed to build a resource of a particular type;	
7	(2) determining whether said first computing entity has stored a resource containing said object; (3) if said first computing entity has stored a resource corresponding to said object, determining if said corresponding resource stored at said first computing entity contains said object; (4) if said corresponding resource stored at said first computing entity does not contain said object, said first computing entity issuing a request for said object;	Document 4, page 2, lines 21-34.
8	(5) responsive to a request for said object from said first computing entity, selecting a software module for building a resource of the type to which said object corresponds; (6) building a resource for containing said object using said selected software module, said resource populated with information defining said resource, but not containing said object; (7) inserting only said object into said resource; (8) transmitting said resource to said first computing entity using said transport mechanism; and (9) providing to said first computing entity said object.	Same recitation as in claim 1. See row 1 above.
9	6. A method for exchanging objects	Same recitation as in claim 1. See

	between two computing entities in an object-oriented programming environment using a transport mechanism in which said data units are contained in files, each file defining a resource, each resource designed to contain a plurality of particular ones of said objects, said method comprising the steps of: (1) providing a resource factory for building resources, said factory including a plurality of software modules for building resources from a data source, each said software module designed to build a resource of a particular type; (2) responsive to a request for an object from a first computing entity, selecting a software module for building a resource of the type to which said object corresponds; (3) building a resource for containing said object using said selected software module, said resource populated with information defining said resource, but not containing said object; (4) inserting said object into said resource; (5) transmitting said resource to said first computing entity using said transport mechanism; (6) providing to said first computing entity said object	row 1 above.

10	(7) providing a reflection adapter factory for populating objects within resources, said factory adapted to provide software modules for populating objects, each said software module designed for an environment corresponding to an object;	Same recitation as in claim 3. See row 3 above.
11	(8) determining whether said first computing entity has stored said property; (9) if said first computing entity has not stored said property, issuing a request for said property	Document 4, page 2, lines 21-34.
12	(10) responsive to said request for said property of said object, selecting a one of said reflection adapters for the environment of the particular property; (11) populating said object with said property; and (12) providing to said first computing unit said property.	Same recitations appear in claim 1. See row 1 above.
13	7. The method of claim 3 wherein said transport mechanism comprises an XML document.	Document 1 discusses using IBM's MOF framework, which deals in XML documents.
14	8. The method of claim 7 wherein said objects comprise Java objects.	All four documents specifically discuss Java objects throughout.
15	9. The method of claim 8 wherein said transport mechanism comprises an XMI document.	Document 1 discusses using IBM's MOF framework, which deals in XML and XMI documents, XMI being a specialized form of XML.
16	10. The method of claim 9 wherein steps (4) and (5) utilize the Meta Object Facility of the Object Management Group specification to read an XMI document.	Document 1 discusses using IBM's MOF framework.

17	11. The method of claim 8 wherein, in step (2), said information defining said resource comprises at least a package object of said resource.	Document 4, page 6, lines 25-50.
18	12. A method for exchanging objects between two computing entities in an object-oriented programming environment using a transport mechanism in which said data units are contained in files, each file defining a resource, each resource designed to contain a plurality of particular ones of said objects, said method comprising the steps of: (1) providing a resource factory for building resources, said factory including a plurality of software modules for building resources from a data source, each said software module designed to build a resource of a particular type; (2) responsive to a request for an object from a first computing entity, selecting a software module for building a resource of the type to which said object corresponds; (3) building a resource for containing said object using said selected software module, said resource populated with information defining said resource, but not containing said object; (4) inserting said object into said resource; (5) transmitting said resource to said first computing entity using said transport mechanism;	Same recitations appear in claim 1. See row 1 above

	(6) providing to said first computing entity said object;	
19	(7) providing a reflection adapter for populating objects within resources, said factory adapted to provide software modules for populating objects, each said software module designed for an environment corresponding to an object;	Same recitations appear in claim 3. See row 3 above.
20	8) determining whether said first computing entity has stored said property; (9) if said first computing entity has not stored said property, issuing a request for said property;	Document 4, page 2, lines 21-34.
21	(10) responsive to a request for said property of said object, selecting a one of said reflection adapters for the environment of the particular property;	Same recitations appear in claim 1. See row 1 above
22	11) determining whether said selected reflection adapter has previously reflected said requested property; (12) if said first computing entity has previously reflected said requested property, populating said object with said property; and (13) providing to said first computing unit said property.	This is the hasReflected tag mentioned in Document 3, lines 31-33.
23	13. The method of claim 1 wherein said data source for building said resources comprises a live system.	The live Java object implementation disclosed in Document 4, page 4, lines 2-14 is just one example of a live system.
24	14. The method of claim 1 wherein said data source for building said resources comprise a database.	Support found in same evidence identified in preceding row.
25	15. The method of claim 1 wherein said data source for building said resources comprises a document in a format other than a format of said transport mechanism.	In Document 4, the "JDOM" reflection mechanism parses a Java source file resource, which is another format than the transport, which is basically XMI. See Document 4, page 6, lines 20-50.

Document 1

line nos.

Java model reflection design

Created By: Scott Rich on 06/01/2000 at 08:00 AM
Category: Miscellaneous

The current beaninfo mechanism provides a mechanism to populate a Java model dynamically by introspecting the live Java class. This is extremely useful for many models which capture metadata which is closely coupled to Java classes. There are two problems with the current approach:

1. The current mechanism use JavaBean introspection to populate the fields of the JavaClass. This isn't ideal for tools other than the VCE where Java reflection or JDOM description would be more accurate.
2. The current mechanism mixes the reflection capability with a proxy capability to introspect classes and interact with instances in remote JVM's. This capability is not generally required by tools other than the VCE.

In order to enable other Java model users to get proper reflection while maintaining the capability for the VCE to use its proxied introspection, we propose to refactor the current beaninfo mechanism. It will essentially be split into two components:

1. Java model reflection: JavaClass will have inherent support for reflection, which will be pluggable.
2. Beaninfo introspection: layered on top of 1.

The current beaninfo mechanism is implemented with a URIParser scheme called "beaninfo". Users can refer to a live class with a URI like "beaninfo://java.lang.String". Today, the document returned from a beaninfo parser contains specialized subclasses of JavaClass (BeanType) and Field (BeanPropertyAttribute) which extend the base types with BeanInfo attributes. The BeanInfo mechanism (the version I have anyways) does not yet introspect methods. The property introspetion is done lazily, presumably so that superclasses don't need to be fully instantiated if their contents is never used.

The Java model will introduce a "java" scheme implemented on top of JDOM or straight reflection. When a user asks for "javabean://java.awt#Button", it will roll over to Java and ask for "java://java.awt#Button." This will ensure that there is a single unique instance of the class around. Java will poof up a JavaClass if needed. The JavaBean mechanism will create a new object which implements JavaClass and another interface JavaBeanClass, maybe called BeanDescriptor. It will populate the beaninfo properties of the JavaBeanClass using bean introspection, and will delegate the JavaClass behavior on to the Java instance. (Joe: We no longer need the extended tool info with this design, do we?) When a user edits a JavaBeanClass, its bean-specific properties can be serialized in a document along with a "java:" reference to the underlying JavaClass.

The proxy behavior which is mixed into the beaninfo mechanism will be moved out to the bean adapters. In order to keep the JavaBeanClass in synch with the JavaClass, Joe would like to have an Observer on the JavaClass. I think this is consistent with our discussions with Frank. Until we get to MOF2, I guess this could be a lightweight Adapter to some empty interface. Although with the delegating design, hopefully there is less synchronization required.

Questions:

1. Does introspection need to be done lazily?
2. Should this go right in the model or on the side?

Line nos.

Document 2

6/7 Foundation meeting

Created By: Scott Rich on 06/07/2000 at 02:38 PM
Category: Miscellaneous

EJB status:
- pretty significant update of Java and EJB models in current build
    - we're dropping a JUnit plugin for testing
- mapping is moving along well, Frank demoed prototype of top-down and bottom-up for EJB<>RDB
- deployment of MOF EJBs to 3.5 format is almost done
    - should be in plug-in form next week
- query writes are done, single-table reads almost done
- Java model introspection and beaninfo rework firing up, hopefully deliver about 2 weeks
- import/export work firing up
- Rational discussion on bridge started, they are about to ship EJB modelling tools

Frank
- rdb mapping dependency for SDK2
- MOF2 major integration point rather than SDK2 driver timeframe, probably +2/3 weeks

Document 3

Line Nos.

Scott Rich /Raleigh/IBM
06/20/2000 06:44 PM
This document expires on
04/22/2003

To ritchie schacher
cc
bcc
Subject Java code

Hey Ritchie,
Here's the Java code as it exists:

It's a little hard to exercise the reflection stuff, but you can do it form the EJB action if you replace it with the following method:

```
/**
 * Perform the appropriate action.
 */
public void actionPerformed(IAction action, Window shell) {
	// No-op for now
	IWorkbench w = EJBPlugin.getPluginWorkbench();
	ISolution s = ((Workbench) w).getOpenSolution();
	IJavaModel javaModel = JavaModelManager.getJavaModel(s);
	IJavaProject[] projects = null;
	try {
		projects = javaModel.getJavaProjects();
	} catch (JavaModelException e) {
		System.out.println("no loaded projects");
		return;
	}
	IJavaProject firstProject = projects[0];
	Factory f = FactoryImpl.instance();
	Document d = f.createDocument("testing.xmi");
	((XMIDocumentImpl) d.getXMIDocument()).addAdaptorFactory(new
JavaJDOMAdapterFactory(firstProject));
	JavaPackage javaLang = f.createJavaPackage();
	javaLang.setName("java.lang");
	JavaClass string = f.createJavaClass();
	string.setName("String");
	javaLang.addContents(string);
//	DebugSupport.inspect(string);
	string.refValue(FactoryImpl.JavaClass_isPublic);
}
```

The blue lines are the interesting ones. Also, I ran into problems trying to make the read adaptors really lazy, due to the handling of default values and the fact that it would have required modifying all the accessors. The adaptor I've built is just like the BeanInfo ones, which hold a "hasReflected" flag, and push settings into the target on teh first request.

Let me know what you think, hope this is helpful.

Scott

Scott Rich
IBM VisualAge Features Development
(919) 254-1943 (tieline 444)
srich@us.ibm.com

java.dat has been deleted (was saved in repository My Attachments Repository ->) from this note on 09 August 2000 by Ritchie Schacher

Document 4

Line nos.

Java reflection design and migration to MOF2

Created By: Scott Rich on 09/21/2000 at 09:34 AM
Category: Miscellaneous

The EJB team has built a MOF Java model which captures the essential features for modelling Java packages and classes. It is initially used by the EJB model to provide a MOF representation of the classes, methods, and fields which are part of the specification of an EJB. The Java model provides the capability to populate a JavaClass by reflecting its features from an existing Java class. This allows the EJB model to refer to existing Java elements using a special HREF notation, without requiring an EJB tool to explicitly create a model of the expected Java class structure. (To avoid confusion, we will refer to the above Java model as "jMOF", and use "Java Model" to refer to the Java model provided in VA/Base as part of the Java IDE.)

Since the Java model has to run in VA/J, in the VA/Base workbench with its Java model, and potentially outside the workbench with the JDK only, the reflection capability of the model is pluggable. Implementations currently exist for reflection using the VA/Base Java Model in the workbench cases, and for using the java.lang.reflect capability provided by the JDK in non-workbench cases.

The jMOF reflection capability is currrently implemented on MOF 1b, and needs to be reworked for MOF2. We'd like to take this time to get feedback from the MOF team on how to best perform this migration, as well as solicit feedback from the extended Eclipse team on the reflection design and capability. Specifically, the VCE team will need to implement their beaninfo introspection capability as a layer on top of our reflection, so we need them to understand and be satisfied by the reflection design.

Since the reflection of Java class structure can be expensive, it was decided that we should be lazy with regards to invoking JDK or Java Model reflection. The design which we chose to implement to perform this lazy reflection was an evolution of the design used by the original beaninfo introspection which Joe and Rich did. In order to take advantage of this capability, clients of jMOF must refer to jMOF elements using a Java URI scheme. This scheme handles a URI which appears as follows: "java: *//com.sparkysoft.mypackage.MyClass*". The URL can be extended with id-refs to class elements such as fields and methods. They are encoded by appending "*#fieldname*" or "*#method.parmTypeName1.parmtypeName2*". Here is an example of an EJB model which refers to jMOF elements:

```
<ejb:ContainerManagedEntity xmi.id="EJB_Customer" xmi.uuid="EJB_Customer"
name="Customer" isReentrant="false" persistentFeatures="Customer_lastname Customer_title
Customer_customerid Customer_firstname" keyFeatures="Customer_customerid">
  <ejb:Entity.primaryKey>
    <java:JavaClass xmi.uuid="CustomerKey" href="java://ITSOEjbs"/>
  </ejb:Entity.primaryKey>
  <ejb:EnterpriseBean.ejbClass>
    <java:JavaClass xmi.uuid="CustomerBean" href="java://ITSOEjbs"/>
  </ejb:EnterpriseBean.ejbClass>
  <ejb:EnterpriseBean.homeInterface>
    <java:JavaClass xmi.uuid="CustomerHome" href="java://ITSOEjbs"/>
  </ejb:EnterpriseBean.homeInterface>
  <ejb:EnterpriseBean.remoteInterface>
    <java:JavaClass xmi.uuid="Customer" href="java://ITSOEjbs"/>
  </ejb:EnterpriseBean.remoteInterface>
  <MOF:Namespace.contents>
    <MOF:Attribute xmi.id="Customer_lastname" xmi.uuid="Customer_lastname"
name="lastname"/>
  <MOF:Attribute xmi.id="Customer_title" xmi.uuid="Customer_title" name="title"/>
  <MOF:Attribute xmi.id="Customer_customerid" xmi.uuid="Customer_customerid"
```

```
        name="customerid"/>
            <MOF:Attribute xmi.id="Customer_firstname" xmi.uuid="Customer_firstname"
        name="firstname"/>
            </MOF:Namespace.contents>
            </ejb:ContainerManagedEntity>
```

The URI parser which is registered to handle this scheme is com.ibm.etools.java.adapters.JavaDocumentParser. It uses a helper class JavaURL which makes it more convenient to construct and parse the above URL's. Here is an example of its usage:

```
        public XMIReference createClassRef(String targetName) {
                XMIReference ref = MOFConstants.mofClassifier.newReference();
                ((RefObjectExtnImpl) ref).setXMIDocument(((XmiObject)
getTarget()).getXMIDocument());
                JavaURL javaurl = new JavaURL(targetName);
                ref.setURL(javaurl.getFullString());
                return ref;
        }
```

In addition, there is an API on com.ibm.etools.java.Factory which can be used to explicitly lookup a Java class by qualified name. Here is an example:

```
        JavaClass myClass =
com.ibm.etool.FactoryImpl.instance().getJavaType("my.package.MyClass");
```

The Java URI parser is specialized to use the jMOF Factory to load the document, which in turn is specialized to return a specialized XMIDocument, a com.ibm.etools.java.adapters.JavaXMIDocument. An important feature of this flow is that the jMOF factory first looks up the URL in the global document cache held by XMIDocumentImpl, via the following code:

```
        JavaXMIDocument xmiDoc = (JavaXMIDocument)
XMIDocumentImpl.getCachedDocument(name);
```

This provides two important functions: first, it is a performance optimization preventing the reflection of a Java Package more than once, second, it allows jMOF or another component to "pre-load" documents which will be frequently referenced. jMOF preloads a document for "java://java.lang" and "java://" which is where the primitive types can be found. The effect of having documents globally cached is that the jMOF document for any package which has already been referenced will always be available. The document itself can then resolve object references within that document, lazily instantiating classes as required. Classes which are already reflected will be "cached" by the document by id and will be returned immediately. This also ensures a single instance of a reflected JavaClass in the model.

(Notes on primitive types:
They can be accessed through the above APIs, here are is an example:

```
    Classifier intType = f.getJavaType("int");
```

You can also refer to a primitive type as follows:

```
    new JavaURL("int");
```

which eventually expands to: "java://#int")

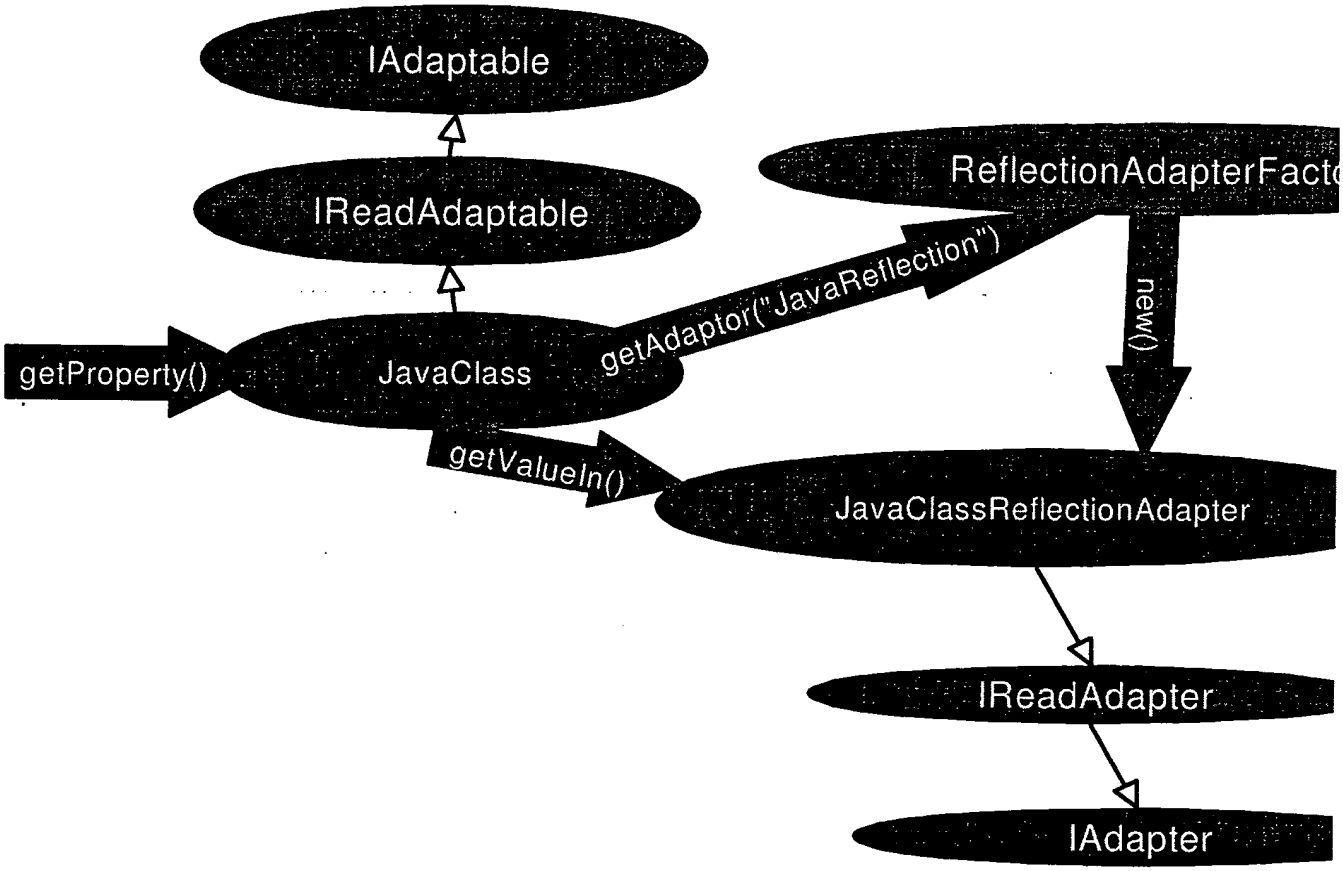
JavaXMIDocument implements the responsibility for interpreting a JavaURL as a reference to an appropriate jMOF element. This is done by overriding the getObject() method to analyze the JavaURL and then instantiating an appropriate jMOF object. The initial access to a package will create the JavaPackage object as the root object of the document. As classes are referenced, they will be added to the package. If a direct reference is made to a class feature before the class, the JavaClass will be created and the referenced feature will be returned from the class. This mechanism is currently always lazy. While resolving references, the existence of a package or class is not enforced. An extension is probably required to allow the client to specify a more strict behavior, where references should fail if a class does not exist.

(flow for resolving a reference:

JavaXMIDocument.createJavaClass()

```
                this=(com.ibm.etools.java.adapters.JavaXMIDocument) JavaDocument(ITSOEjbs)
        JavaXMIDocument.getJavaClass(String)
                this=(com.ibm.etools.java.adapters.JavaXMIDocument) JavaDocument(ITSOEjbs)
                typeName=(java.lang.String) CustomerKey
        JavaXMIDocument.getJavaObject(String)
                this=(com.ibm.etools.java.adapters.JavaXMIDocument) JavaDocument(ITSOEjbs)
                keyValue=(java.lang.String) CustomerKey
                className=(java.lang.String) null
                attributeName=(java.lang.String) null
                result=(com.ibm.mof.mof13.ref.extn.RefObjectExtn) null
                index=(int) -1
        JavaXMIDocument.getObject(Object)
                this=(com.ibm.etools.java.adapters.JavaXMIDocument) JavaDocument(ITSOEjbs)
                key=(java.lang.Object) CustomerKey
                javaObject=(com.ibm.mof.mof13.ref.extn.impl.RefObjectExtnImpl) null
        XMIDocumentImpl.get(String, String)
                this=(com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl)
com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl(c:\testing\abaxx\meta-data\EJBModel.xmi)
                documentURL=(java.lang.String) java://ITSOEjbs
                resourceName=(java.lang.String) CustomerKey
                doc=(com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl) JavaDocument(ITSOEjbs)
        XMIDocumentImpl.get(URL)
                this=(com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl)
com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl(c:\testing\abaxx\meta-data\EJBModel.xmi)
                url=(com.ibm.xmi.xmi11temp.impl.URL) URL(java://ITSOEjbs#CustomerKey)
        XMIDocumentImpl.getMOFObject(String)
                this=(com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl)
com.ibm.xmi.xmi11temp.impl.XMIDocumentImpl(c:\testing\abaxx\meta-data\EJBModel.xmi)
                urlString=(java.lang.String) java://ITSOEjbs#CustomerKey
                startingObject=(com.ibm.mof.mof13.ref.extn.impl.RefObjectExtnImpl) null
                url=(com.ibm.xmi.xmi11temp.impl.URL) URL(java://ITSOEjbs#CustomerKey)
        XMIReferenceImpl.getTarget()
                this=(com.ibm.mof.mof13.extn.impl.XMIReferenceImpl)
MOFReference('java://ITSOEjbs#CustomerKey')
        MOFLink.getObject()
                this=(com.ibm.mof.mof13.extn.impl.MOFLink)
primaryKey(MOFReference('java://ITSOEjbs#CustomerKey'))
)
```

The JavaClass is the first jMOF object which really has reflection behavior. As soon as a JavaClass is asked for any reflectable feature, it will invoke its reflection capability to populate its features. (The features name, id, uuid, and container are pre-set in order to make it possible for reflection to run.) The design for jMOF reflection is as follows:



A pair of new interfaces has been introduced to extend the current adapter functionality. Our thought was that the current adapter design supported the "write" half of adpater behavior, signalling observers when new values were written to an object, ultimately via teh generic set accessor: refSetValue(). Our extensions allows adapters to participate in the "read" portion of a MOF object's behavior. That is, to be hooked into the get accessor: "refValue()". So IAdapter was extended by IReadAdaptor which adds the main path: getValueIn(anObject, anAttribute). Of course it got more complicated, and we ended up extending IAdaptable with IReadAdaptable, specifically to allow objects using read adaptors to re-implement some primitive getValue() behavior to break recursion. Read adaptors are created by adaptor factories just like write adaptors, and they are differentiated by their adaptor type, in this case "JavaReflection". The kind of jMOF reflection which gets applied in a given application is determined by which adaptor factory gets activated. jMOF FatcoryImpl has some logic to detect whether Java Model is available, in which case it sets the (mis-named) JavaJDOMAdapterFactory to be the active "JavaReflection" adapter factory.

```
protected IAdaptorFactory getAdaptorFactory() {
    IAdaptorFactory factoryInstance = null;
    try {
            Class factoryClass = null;
            if (workbenchIsRunning())
                    factoryClass =
Class.forName("com.ibm.etools.java.adapters.JavaJDOMAdapterFactory");
            else
                    factoryClass =
Class.forName("com.ibm.etools.java.adapters.JavaJDKAdapterFactory");
```

```
                factoryInstance = (IAdaptorFactory) factoryClass.newInstance();
        } catch (Exception e) {
                // Reflection or instantiation problems.
                // OK, can't do Java Model reflection
        }
        return factoryInstance;
    }
```

Because we wanted to use our read adaptors for lazy initialization of our JavaClass fields, and because the generated methods no longer pass through refValue() is there is a cached value, we have to override all of the relfectable property get-methods in JavaClassImpl to perform reflection. These methods all follow variations on the following form:

```
    protected IBMMOFSuperclass getSuperclass() {
        if (superclass == nullSuper) {
                // trigger reflection and superclass should be updated
                // this is a somewhat slimy assumption.
                getReadAdaptorValue(MOFConstants.mofSuperclassRole);
        }
        return super.getSuperclass();
    }
```

Getters for Enumeration and primitive-typed properties use static helpers on Reflection adapter to handle their more complex reflection, like so:

```
    // Enumeration Constants for following are defined in interface : TypeKind
    public int getKind() {
        return ReflectionAdaptor.getEnumValue(this, FactoryImpl.JavaClass_kind);
    }
```

Here is an example flow for a reflected property:

```
        JavaClassJDKAdaptor.reflectValues()
                this=(com.ibm.etools.java.adapters.JavaClassJDKAdaptor)
com.ibm.etools.java.adapters.JavaClassJDKAdaptor@7d1f
        JavaClassJDKAdaptor(ReflectionAdaptor).reflectValuesIfNecessary()
                this=(com.ibm.etools.java.adapters.JavaClassJDKAdaptor)
com.ibm.etools.java.adapters.JavaClassJDKAdaptor@7d1f
        JavaClassJDKAdaptor(ReflectionAdaptor).getValueIn(RefObjectExtnImpl, RefObject)
                this=(com.ibm.etools.java.adapters.JavaClassJDKAdaptor)
com.ibm.etools.java.adapters.JavaClassJDKAdaptor@7d1f
                object=(com.ibm.mof.mof13.ref.extn.impl.RefObjectExtnImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
                attribute=(com.ibm.mof.mof13.ref.RefObject)
com.ibm.mof.mof13.extn.impl.AssociationEndExtnImpl(AssociationEnd(contents))
        JavaClassJDKAdaptor.getValueIn(RefObjectExtnImpl, RefObject)
                this=(com.ibm.etools.java.adapters.JavaClassJDKAdaptor)
com.ibm.etools.java.adapters.JavaClassJDKAdaptor@7d1f
                object=(com.ibm.mof.mof13.ref.extn.impl.RefObjectExtnImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
                attribute=(com.ibm.mof.mof13.ref.RefObject)
com.ibm.mof.mof13.extn.impl.AssociationEndExtnImpl(AssociationEnd(contents))
        JavaClassImpl.getReadAdaptorValue(RefObject)
                this=(com.ibm.etools.java.impl.JavaClassImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
                attribute=(com.ibm.mof.mof13.ref.RefObject)
com.ibm.mof.mof13.extn.impl.AssociationEndExtnImpl(AssociationEnd(contents))
```

```
        JavaClassImpl.getContentsEnum()
                this=(com.ibm.etools.java.impl.JavaClassImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
        JavaClassImpl(NamespaceExtnImpl).getContents()
                this=(com.ibm.etools.java.impl.JavaClassImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
                c=(com.ibm.mof.mof13.ref.Collection) []
        JavaClassImpl(MofClassExtnImpl).findElementsByType(MofClass, boolean)
                this=(com.ibm.etools.java.impl.JavaClassImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
                ofType=(com.ibm.mof.mof13.MofClass)
com.ibm.mof.mof13.extn.impl.MofClassExtnImpl(Method)
                includeSubtypes=(boolean) false
                result=(com.ibm.mof.mof13.ref.Collection) []
        JavaClassImpl.listMethod()
                this=(com.ibm.etools.java.impl.JavaClassImpl)
com.ibm.etools.java.impl.JavaClassImpl(String)
        Object.Doit()
                string=(com.ibm.etools.java.JavaClass) com.ibm.etools.java.impl.JavaClassImpl(String)
    )
```

There are two base classes for jMOF reflection adapters which provide helpers and a template pattern for implementing additional reflection adapters using those Java reflection techniques . Here is an example of e reflection adapter implementation, this is the main method of JavaClassReflectionadapter, reflectValues():

```
        /**
         * reflectValues - template method, subclasses override to pump values into target.
         * on entry: name, containing package (and qualified name), and document must be set.
         * Return true if successful
         * JavaClass adaptor:
         *      - set modifiers
         *      - set name
         *      - set reference to super
         *      - create methods
         *      - create fields
         *      - add imports
         */
        public boolean reflectValues() {
                if (getSourceType() != null) {
                        setModifiers();
                        setNaming();
                        setSuper();
                        setImplements();
                        addMethods();
                        addFields();
                        //      addImports();
                        return true;
                }
                return false;
        }
    )
```

Within the reflection implementation for JavaClass, we continue to be lazy about reflection the details of fields and methods. The field and method features fo JavaClass are populated with partial JavaMethod and JavaField objects, which will later reflect their contents if required . Their reflection is implemented using the same pattern as class. The critical information for a partial object is that is have enough context

information to locate its corresponding Java artifact and continue reflection. This generally means that the container (JavaClass or JavaPackage) must be set, and the name or id of an object must contain enough information to uniquely identify the contained element. For a method, for example, the id must contain all of the parameter type names which must be combined with the name to form a unique method signature.

Other applications :

As something of an experiment, we used this same ReadAdapter pattern for implementing some other transformations. In the J2EE world, there are several standard deployment descriptors which are XML documents describing EJBs, web applications, and enterprise applications. We need to parse these and populate a corresponding MOF model to allows the tools to manipulate them. We built read adapters for the main objects in these models, and those read adapters perform the translation from XML elements to MOF objects and properties. The read adapters are initialized with the root TxElement representing the portion of the XML document which is relevant to initializing a MOF object. We have enabled a switch to have reflection occur eagerly or lazily. If it turns out the eager reflection is the only useful case, we may back away from using this pattern.

Migration to MOF 2:

Our migration to MOF2 is currently blocked by our lack of understanding of how to map the reflection implementation to MOF2. I think we need a better understanding of (at least) the following issues:

- XMIDocument seems like the wrong place to hang the package reflection and JavaURL resolution. Do we need a specialized Resource or Resource+Extent to implement this?
- Where is the document cache previously owned by XMIDocumentImpl? Do we need to create an XMIResources, what is the scope of that?
  - Do we create a specialized XMIResources which knows to create our JavaXMIResource?
- We use XMIReference all over the place, and we're not sure what we use now for proxying.
- How does our adapter extension fit into the new Notifier/Observer model?

.....